2008 International Conference on Software Testing, Verification, and Validation

Test Data Provision for ERP Systems

Sebastian Wieczorek, Alin Stefanescu
SAP Research CEC Darmstadt
Bleichstraße 8, D-64283 Darmstadt
sebastian.wieczorek@sap.com,
alin.stefanescu@sap.com

Ina Schieferdecker
Fraunhofer Institute for Open
Communication Systems (FOKUS)
Kaiserin-Augusta-Allee 31, D-10589 Berlin
ina.schieferdecker@fokus.fraunhofer.de

Abstract

Software development and testing of Enterprise Resource Planning (ERP) systems demands dedicated methods to tackle its special features. As manual testing is not able to systematically test ERP systems due to the involved complexity, an effective testing approach should be automated, also requiring that the appropriate test data has to be provided alongside. In this paper we identify four main challenges regarding the provision of test data for automatic testing of ERP software: system test data supply, system test data stability, input test data constraints and test data correlation. Several possible solutions to these challenges are discussed. We conclude with an outlook to possible research activities.

1. Introduction

In the eve of industrialization of software development, automation is playing a major rôle. As a way to validate the software quality, software testing puts a lot of effort in automation as well. Over the time, we have seen manual testing enhanced by capture and replay capabilities, then test scripting languages were employed for automation, especially for the regression testing [8]. Test data is very important if we want to have a seamless test automation process. In this paper we will discuss several problems regarding ERP test data provision that can be identified in the area of functional testing of ERP systems.

Enterprise Resource Planning (ERP) software [14,22] is built to support business processes for whole companies. SAP, the world's leading provider of business software, has a large history of developing ERP systems. Such software systems are very complex: e.g. SAP R/3 consists of over 250 million lines of code [15] being considered as one of today's largest software systems. Not only their size but also the huge data volume they are managing makes ERP systems very complex. ERP software integrates many organizational parts and functions into one logical software system, posing its own challenges to testing [9].

When it comes to the data used during testing, questions as below need to be properly addressed: "Where will the test data come from, especially the initial test data pool? How will one produce new data for transactions that require unique data? How one ensures the consistency of master, transactional, respectively test data?" All these questions relate to what we call *test data provision*.

In this paper, we will focus on tests carried out during the product development phases. Functional testing at customer side, e.g. after the customers' IT department added or changed features to an SAP product, addresses different additional concerns like test data confidentiality and limited system access. These features are equally relevant in practice but will not be addressed here. The issues concerning test data discussed in this paper are however relevant for customer testing, too [9].

The key contributions of this paper are a classification of problems related to test data for ERP systems including an identification of test data constraint types, an analysis of existing pragmatic solutions and a listing of interesting research questions. The paper is structured as follows. First, we present some preliminary notions and related work in Section 2. We continue in Section 3 describing the different ERP test data and the relations between them. Section 4 consists of a collection of identified challenges and Section 5 presents our future research plans.

2. Preliminaries and related work

In this section we present concepts and methodologies from the literature to which we will refer in the next sections.

978-0-7695-3127-4/08 $25.00 © 2008 IEEE
DOI 10.1109/ICST.2008.53

A test case describes the operational steps through which a certain functionality of property of the System-Under-Test (SUT) is validated. Following [10], a test case consists usually of the concatenation of four procedures:

(1) *Preamble* (or setup): Sets the test up, getting the SUT into the correct state to run the test;
(2) *Body*: Executes a certain scenario or sequence of steps described by the test case in the SUT;
(3) *Observation* (or verification): Checks and evaluates the test results and
(4) *Postamble* (or teardown): Takes the SUT back in some standard state, where the next test can run.

Regression testing is used to ensure the quality of software systems produced in several development cycles, by checking that additional code and changes do not affect the functionality already implemented and that the requirements are always satisfied. It consists of a set of test cases that are executed regularly at every stable stage of the development cycle. These test suites should be reproducible and this requires a constant initial system state at the beginning of a test run together with appropriate test data. Such requirement can be easily satisfied by stateless systems or systems with a constant state at the end of each session, e.g. protocol machines. In this case, test cases can be annotated with static and concrete test data. Some systems can be forcibly brought into a defined initial test state using a preamble procedure. For other systems however the requirement of starting a test in a constant initial state cannot be met. For example it is practically impossible to reset ERP systems because the effort is simply too high even in a developing stage and for a running ERP system this might even not be allowed, e.g. due to legal obligations. This brings into question the testability [21] of such systems, i.e. to which degree such systems allow to define and execute an effective testing process. If we take for instance the controllability and observability features, they need a careful analysis on how input test data produces, respectively affects the output data of a system. Moreover, issues regarding test data collection, dependability, and reliability need also to be evaluated for ERP systems.

As software systems are becoming more and more complex, new levels of abstractions are introduced both in software development and testing. Model-based testing (MBT) [19] is a kind of black-box testing [3] that uses structural and behavioral models, described for instance by UML diagrams, to automatically generate abstract test cases, thus automating the test case design process. MBT is gaining its momentum together with the model driven software development (MDSD) methodology. MBT test case generators aim to cover by test cases model features, e.g. all states in an UML state machine, or data features, e.g. all boundary values. As most system models are not complete in terms of data modeling, (due to complex data constraints), the test generators produce abstract test cases which have to be further refined by adding concrete data. Therefore, it is important to have a good modeling framework for test data, such that the appropriate test data are eventually generated along with the generated test cases.

Test data constraints have to be considered during test design when annotating test cases with data. The reason is that system state changes do not usually depend only on the performed actions, but also on the input data and the actual internal state, including internal data and state variables. The current test data constraints focus on transactional data relations during the run of one test case only. Because it was relatively easy to annotate test cases with concrete data using the mentioned restriction, test data modeling was not highly prioritized and so a consolidated approach to link data models and behavior models is still missing [17]. To the best of our knowledge, a classification of different test data constraints especially for ERP systems is missing.

Data modeling is usually applied in the field of database layout planning and management. Relational modeling, e.g. using the Entity-Relationship Model, is the most common way to describe database schemas and languages like SQL have been invented to retrieve the data from such relational databases. The object-relational database model recently came to prominence, reflecting the paradigm shift in programming and providing an object oriented data structure in databases. Related modeling techniques use object-oriented models, enhanced by constraints. The most popular approach utilizes the OMG's UML standard and OCL constraints. A discussion on the adoption of the UML/OCL approach and its possible challenges can be found in [1]. For ERP system tests, data modeling has to be adapted as the data layout inside the system is distributed and must address specific challenges as the ones presented in this paper.

Ontologies in computer science represent a set of concepts within a domain and the relationships between those concepts. Instances of the concepts as well as domain rules are also part of an ontology. Unlike data models, ontologies are supposed to be independent of a particular application of the described domain [18]. Given their genericity, ontologies can be applied to a limited degree in the highly customized ERP systems.

Test data generation has been discussed in the context of automated testing from the very beginning. For white-box testing, approaches like symbolic execution, actual execution, and random testing [7] are used. They are based on code analysis or code execution tracing to find value sets needed to execute predefined paths through a system under test. For black-box testing and in particular MBT, test data can be generated from several types of specifications like finite state machines, pre/post models, or UML transition-based models [19,13,5]. Most of the research addresses the test data generation for each test case independently and less the global consistency of different test data.

The problem of test data modeling has been tackled in different ways, by approaches like boundary analysis [11], domain analysis [3] or the classification tree method [6], but these are not very well integrated with the behavior models [17]. Test data modeling using the UML/OCL data modeling approach is a promising alternative [1,5,2]. We believe however that the problems of incomplete data constraint definitions and constraint solving of very complex systems such as ERP systems need further investigations. Our paper takes the first steps into this direction by identifying the main challenges and types of constraints symptomatic for ERP systems.

3. An introduction to ERP data

In the ERP world, data can be interpreted from two perspectives: from a business, respectively from a technical perspective.

Business view on ERP data. From a business point of view, data is divided into master data and transactional data:

- Master data represents static data that remains valid over a period of time and is used in several use case scenarios. For example supplier information (such as name and address) or product information (such as size and description) is stored once, seldom changed and can be inserted automatically during several transactions.
- Transactional data in contrast is short-lived, used only for a specific transaction and can always be related to master data. For example ordering of a product will be processed in a sales order transaction. Information like the quantity of products or the delivery deadline is individual for the order and hence transactional data.

Technical view on ERP data. From a technical perspective the distinction between master data and transactional data is less relevant. All transactions in a system have to be stored in databases and therefore master data tables have primarily the purpose of eliminating redundancies. In the above example of a sales order only the transactional data is saved explicitly while master data is referenced. More important from a technical perspective is the distinction between user generated and automatically derived data for a transaction. Note also that transactional data is not only generated by user input but also might be derived automatically (e.g. the current date) or from a prior transaction. For example the quantity of a product in a sales order might be determined by the current need for production or a request from a customer. Therefore, a technical distinction between system data and input data will be used (see also Figure 1):

- System data serves as the applications internal data set. It is stored in a database that is directly accessible from the application. Access from the outside is usually very limited. System data consists of both master and transactional data.
- Input data is all information that has to be provided from outside by users or external components during execution and cannot be derived automatically. The input data may be master or transactional data.

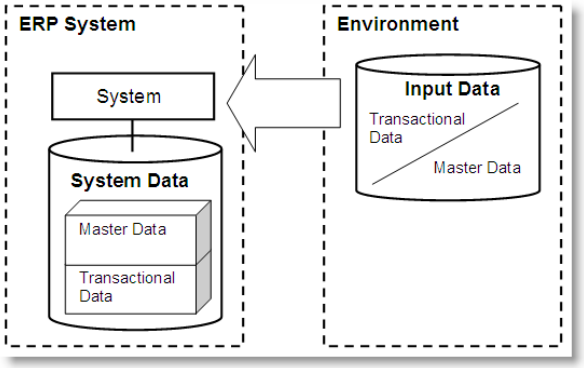

Figure 1. An overview of ERP data: system and input data

The motivation for such a classification is that data which is stored in the system can be considered as consistent and conform to predefined technical and business constraints (as explained above this also includes saved transactional data). The assumption only holds if the system correctly implements all consistency checks and constraint determinations. Consequently, the runtime system only has to check those consistency rules and constraints during a transaction, which are connected to the user generated data input. This is important because the number of transaction critical constraints that have to be checked

before saving any data must be reduced to a manageable subset in order to be enforceable in a reasonable, i.e. user convenient, timeframe.

Consistency of ERP data. Data consistency (e.g. concerning technical, business, and standardization constraints) can be enforced at different levels in the system. Common database systems provide means to define simple to very complex and even dynamic constraints on data. The constraints are defined using field properties or guards which are checked every time the associated data is changed. Conformance to the ACID concept (Atomicity, Consistency, Isolation, Durability) further allows rollbacks of whole transactions, even if constraints are violated at the last processing step. However constraint validation on data in ERP systems is usually implemented outside the database system as early as during input data processing (close to the UI layer). Its main objective is to resolve constraints (or rollback transactions) quickly and without blocking additional resources. While consistency checks on the application level is scalable and fast, a constraint violation found on the database level itself results in a rollback that has to be communicated back to the user interface through all application layers in between and hence can be seen as a performance bottleneck for ERP systems. Shifting constraint handling to the application also allows ERP system designs that are independent of the database implementation. This is due to the fact that database vendors each have their individual language to define data constraints. Correct implementation of data constraint handling and checking therefore is one of the major objectives of ERP application testing.

4. Challenges posed by ERP test data

This section gives an overview of test data characteristics for ERP regression testing. Issues regarding system test data and input test data are separately investigated.

As sketched in Figure 2, the black box testing of an ERP system needs a test execution environment (sometimes called a test context) that groups a suite of test cases. In the case of MBT, this will contain information on the structure and behavior of the system. The execution environment must of course be stable and deterministic in order the achieve repeatability of the test process. Each test case has specific test data associated with each of the test steps. The quality of test data must ensure a good coverage and error detection capabilities. Since such properties are already well studied in the literature [3,8], in this section we will focus on the existing constraints between various data manipulated during testing. Addressing such test data constraints boils down to the four identified properties from Figure 2, namely *system test data supply*, *system test data stability*, *input test data constraints* and *test data correlation*. These will be introduced and discussed in the following two subsections, which are partitioning these properties into system test data or input test data related.

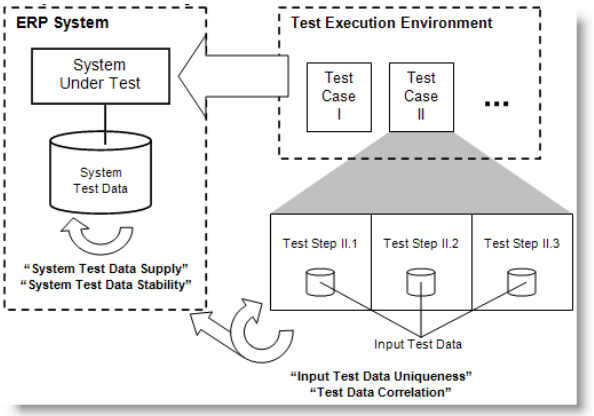

Figure 2. Properties for ERP test data considered in this paper.

4.1 System test data

System data is a necessary ingredient for testing since internal data is the base of any ERP system and will most likely be processed during any execution. Two special subjects regarding system data for regression testing have to be considered:

(a) *system test data supply* and
(b) *system test data stability*.

The details are given below.

(a) System test data supply. In ERP systems, providing master data as system data for later interactions is an important user scenario. For testing, providing an initial system data could be either part of each test relying on system data or could be done during a general testing preparation.

In the first case, all tests should be able to run on empty systems, initializing and storing needed system data in the preamble phase. Such approach is unfortunately infeasible for ERP testing. A simple employment process in a Human Resources (HR) module shows the practical weakness. To hire someone, an open position has to be entered to the system. The position demands other information like the unit to which it is assigned. The unit needs information about its manager and the company it belongs to, among others. Further iterations lead to the creation of a vast amount of master data even for a simple test run. Other processes (e.g. creating a report

for all employment processes) additionally need saved transactional data inside the system. There are usually numerous internal data dependencies such that most test cases would be forced to set up an unmanageable amount of master data in the preamble phase in order to be executable.

Consequently ERP testing demands the insertion of common test data to the system during the testing preparation, which then would be used during the test execution. A first solution is to write the data directly into the database, but this is difficult, as it demands to either manually or automatically enforce system data consistency during this process. For the manual task the complexity of data relations is too high, whereas for the automated task, consistent data insertion would mean to re-implement the expensive system data constraint checking and solving mechanisms.

A second more realistic solution is to fill the empty system with common test data by using the application and hence the implemented mechanisms enforcing system data consistency. The procedure itself therefore can be seen as a set of fundamental test cases (that can consequently be modeled and generated using MBT). Even though this approach seems to be much more feasible, problems still arise. First, using an untested system to generate a master data stack is error prone and hence the quality of master data will be poor. Furthermore in early production stages mechanisms needed for data insertion might not be fully implemented. Despite the mentioned problems, providing common test data as described by the latter solution is the most practicable solution so far and widely used at SAP.

(b) System test data stability means keeping the core system data unchanged. It is strongly connected to the requirement of regression testing: 'always execute a test case on the same system state'. If there is a common set of system test data which, as argued above, should be provided initially, it should not be changed by any test in order to grant repeatability. This is a very expensive requirement for regression testing in ERP systems, because changing of system data is part of the common ERP functionality. System data will be consumed by some tests, e.g. when a process to dismiss an employee is tested: each time an employee is dismissed the system data will be changed and hence the entity cannot be used for employee related transactions like promotion any more. Other tests might alter common master data unintentionally due to implementation faults in either the SUT or the test case itself.

Obviously altered common test data proves to be problematic for other test cases depending on them. Finding out whether a failed test was caused by a faulty implementation or altered master data is hard to decide and a failed test even might occur randomly e.g. depending on the execution order of test cases. Apart from the described technical difficulties, test cases implying a fault because of altered system data also negatively affect the tester's motivation.

System data access rules, preventing the alteration of common test data may be a solution but enforcing write protections during test execution will cause a difference in behavior of the SUT compared to the delivered system. Hence positive test results are not guaranteeing the absence of errors in the delivered system any more.

Another solution is to bind the concrete system data to abstract test cases during runtime. In this case, rules defined e.g. in OCL could be generated together with the test cases, allowing the test execution system to search for compliant system data and to assign it just before or even during test execution. To determine the current system data state and binding suitable data to the abstract test cases is however too complex in practice. Furthermore, observing the whole system state is (if ever) not possible until very late stages, thus preventing such a testing strategy during most of development period.

Another strategy to supply test data stability is the cloning of master data tables in an initial system state (prior to test execution) to ease and speed up regular data resets, e.g. once a week. Shorter periods are usually impracticable because of the copying costs time in which development and testing has to pause and demands additional manual work. However directly copying master data will always result in the loss of stored transactional data, as former references and relations will be destroyed. Also structural changes of the master data (e.g. adding a field for the gender at the personal data) which are carried out frequently during development phases can result in the invalidation of the master data clone. Automatic reset and re-provision of system data as described in the previous section then is the only way to solve the problem. Nevertheless such a system test data reset takes about half a week for some SAP ERP applications, even though it is nearly fully automated. Therefore the usage is very limited and so the general test data stability remains a difficult problem.

4.2 Input test data

Until now only the test data inside the system has been discussed but in order to test ERP applications also data from the outside has to be provided for the test runs. Mostly this input data will be transactional but in order to test master data modification both transactional and master data might have to be added to

test cases. Similar to system data, the general feature that makes it hard to deal with input data is the complexity of its associated constraints.

In the context of ERP system testing the input test data relations can be classified in two groups:

(a) *input test data constraints* describing the correlations inside a test case that are unrelated to the system data. These constraints can be further refined as follows:
  (a.1) *syntactic input data constraints*
  (a.2) *intra test case constraints*, and
  (a.3) *contextual input data constraints*

(b) *test data correlation* describing the relation between system and input data.

Figure 3 is used to illustrate the mentioned constraints. They are explained in detail below.

(a.1) Syntactic input data constraints. Every ERP system has syntactical constraints on input data, concerning for instance data types and ranges. In the test case from Figure 3 positive system response depends on the usage of the correct integer range for the Product Id. In contrast a value outside the range should result in a system error message.

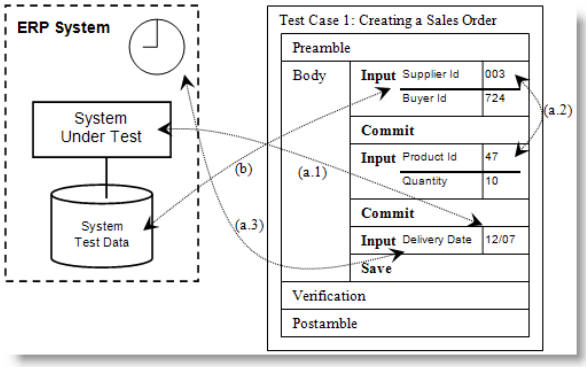

Figure 3. Illustration of different types of input test data constraint

(a.2) Intra test case constraints. Not only in MBT but in every black box testing approach test cases describe an interaction sequence with the SUT using only interfaces, which are accessible from the outside. The correct reaction of the system might not only depend on syntactically correct input values but also on the semantical relation between the data used for different steps of the test sequence. In the example from Figure 3 the system should either react with a success notification or an error message depending on the fact whether supplier #003 is able to provide product #47.

(a.3) Contextual input data constraints. Also the application context might enforce constraints on the test data. In Figure 3 the validity of the used delivery data depends on the current time in the SUT. Other contextual constraints are for example input data restrictions depending on user roles or business configurations.

(b) Input data vs. system data correlation. More complicated constraints are those relating input data to system data. Depending on the system data observability, it might even be impossible to determine in advance whether a certain input value should trigger a positive or negative system response, and hence the satisfaction of constraints might become nondeterministic. In the test case of Figure 3 for each Id (Seller Id, Buyer Id, Product Id) a valid master data entry has to be present inside the system data to be able to successfully process the sales order. However if it is not possible to observe system data during test execution an unambiguous test oracle for the SUT response is impossible to provide. Also the absence of specific system data belongs to this category as master data inside the system often has the restriction to be unique. Hence input of already existing master data might result in different system behavior than the input of unique data. This issue especially becomes prominent in regression tests where the provision of unique input test data might be problematic.

5. Conclusions and future research

In this paper special characteristics of test data in ERP systems have been described. Apart from the complexity of test data constraints for very large systems, also test data supply and stability have been identified as major issues. We will continue to investigate methods responding to each of the identified problems and strategies. Most likely categorization of test cases (e.g. system data consumer / system data dependent / system data independent) and then an orchestration of individual strategies will be part of such a solution.

Our research plans will include the modeling of test data for large ERP systems in the context of MBT and its annotation with constraints, most probably in a UML/OCL context. Here especially the automatic generation of preambles and postambles as well as the application of MBT for negative testing are interesting research topics. Another emphasized difficult aspect is how to provide data for regression testing when the system under test cannot be easily reset. Moreover, MBT may come in two different flavors: online and offline testing [20]. For online MBT, the generation of test cases is dynamic, i.e. the set of test cases already

generated and their result on the SUT impacts on the choice of the next generated test case. In this area we also see a challenge to perfect a dynamic test data generation approach able to cope with unpredictable states and test data sets.

The next-generation product of SAP for mid-market, SAP Business ByDesign[1], will employ data types on a generic level (core components) and data types for specific vertical industry [16]. The Business ByDesign business objects, defined in the Enterprise Service Repository (ESR), are trees of business object nodes. A business object node is structurally defined by a Global Data Type (GDT). In other words, a business object is a structured set of GDTs. As a result the data structures and layouts used inside ByDesign are very diverse and comprise complex associations and interdependencies. We plan to validate our future research prototypes addressing test data constraints internally on a ByDesign system and to compare it with the existing internal test tools

Acknowledgements
This work was partially supported by the EU-funded project Modelplex [12].

6. References

[1] T. Baar: Experiences with the UML/OCL-approach in practice and strategies to overcome deficiencies, In Proc. of Net.ObjectDays2000, Net.ObjectDays-Forum, 2000; pp. 192-201.

[2] H. Balsters: Modelling database views with derived classes in the UML/OCL-framework. In Proc. of UML'03, LNCS 2863. Springer, 2003, pp. 295-309.

[3] B. Beizer: Black-Box Testing: Techniques for functional testing of software and systems. John Wiley & Sons, Ltd., 1995.

[4] M. Benattou, J.-M. Bruel, and N. Hameurlain: Generating test data from OCL specifications. In Proc. of the ECOOP'2002 Workshop on Integration and Transformation of UML models (WITUML'2002), 2002.

[5] L.C. Briand, J. Cui, and Y. Labiche: Towards automated support for deriving test data from UML statecharts. In Proc. of UML'03, LNCS 2863, Springer, 2003, pp. 249-264.

[6] Z. R. Dai, P. H. Deussen, M. Busch, L. P. Lacmene, and T. Ngwangwen: Automatic test data generation for TTCN-3 using CTE. In Proc. of ICSSEA'05. CMSL, 2005.

[7] J. Edvardsson: A survey on automatic test data generation. In Proc. of 2nd Conf. on Computer Science and Engineering, ECSEL, October 1999, pp. 21–28.

[8] M. Fewster, and D. Graham: Software test automation. ACM Press, 1999.

[9] M. Helfen, M. Lauer, and H. M. Trautwein: Testing SAP solutions. SAP Press, 2007.

[10] ISO/IEC 9646-1. Information technology – open systems interconnection – conformance testing methodology and framework, Part 1: General concepts. International Standards Organization, 1994.

[11] N. Kosmatov, B. Legeard, F. Peureux, and M. Utting: Boundary coverage criteria for test generation from formal models. In Proc. of ISSRE'04, IEEE Computer Society, 2004, pp. 139-150.

[12] Modelplex: MODELling solution for comPLEX software systems. 6th Framework Programme EU Project. Reference: IST 34081 (IP). http://www.modelplex.org.

[13] J. Offutt; S. Liu, A. Abdurazik, and P. Ammann: Generating test data from state-based specifications. In Software Testing, Verification and Reliability 12(1). John Wiley & Sons, Ltd., 2003, pp. 25-53.

[14] D. E. O'Leary: Enterprise Resource Planning systems. Systems, life cycle, electronic commerce and risk. Cambridge University Press, 2000.

[15] G. Pike: Supporting business innovation while reducing technology risk. SAP Insight, June 2006. Online at: http://www.sap.com/services/programs/pdf/BWP_Supporting_Business_Innovation.pdf

[16] H. Plattner: Trends and Concepts in the Software Industry (part I). Lecture notes from Hasso Plattner Institute. Summer Term 2007. Webpage (visited January 18, 2008): http://epic.hpi.uni-potsdam.de/Home/TrendsAndConceptsI2007.

[17] I. Schieferdecker: Modellbasiertes testen. In OBJEKTspektrum, vol. 3, 2007.

[18] P. Spyns, R. Meersman, and M. Jarrar: Data Modelling versus Ontology Engineering. SIGMOD Record 31(4), 2002, pp. 12-17.

[19] M. Utting, and B. Legeard: Practical model-based testing, a tools approach. Morgan Kaufmann Publishers, 2007.

[20] M. Utting, A. Pretschner, and B. Legeard: A taxonomy of model-based testing. Technical Report 04/2006, Department of Computer Science, The University of Waikato (New Zeeland), 2006.

[1] http://www.sap.com/solutions/sme/businessbydesign

[21] J. M. Voas, and K. W. Miller: Software Testability: The New Verification. In IEEE Software 12(3), 1995, pp. 17-28.

[22] M. Al-Mashari, Abdullah Al-Mudimigh, and M. Zair. Enterprise resource planning: A taxonomy of critical factors. In European Journal of Operational Research 146 (2), 2003, pp. 352-364.